Неблокируемые туннели

Пиринговые сети (например IPFS) позволяют построить туннель между двумя Proxy-серверами через много разных компьютеров. Proxy-серверы можно написать на Java (для Android-смартфона) и Python (VPS). Либо Python-Proxy на ПК пользователя и Python-Proxy на VPS. На VPS можно использовать и PHP-Proxy, но особого смысла по сравнению с Python нет.

Туннель через IPFS позволяет просматривать статические веб сайты. Сайты с большим количеством скриптов будут доступны с трудом, либо вовсе не доступны.

Решение может быть следующее: разместить на VPS эмулятор браузера, имитировать просмотр динамических страниц веб сайта, формировать статическую версию веб страницы и её отправлять пользователю через IPFS.

Но в этом случае есть риск, что имитатор браузера на VPS будет воспринят сайтом как бот и не получит контента. Чтобы преодолеть это, нужно передать через Proxy 1 в IPFS туннель, из IPFS туннеля в Proxy 2 и от Proxy 2 к сайту реальный запрос пользователя.

Задача вполне решаемая и реализованная на Python.

Такое туннелирование позволяет просматривать сайты за очень жёсткими ограничениями. Но такая технология не обеспечивает интерактивности. Т.е. нельзя логиниться, заполнять поля, писать сообщения. Реализовать такие опции можно, но путём значительного усложнения или разработки кастомных версий для конкретных сайтов.

Принципиально можно использовать сеть Bit Torrent, I2P (Invisible Internet Project) или Mythos. Но ограничений там не мало и для эффективной разработки кода надо использовать AI Coding (Vibe Coding.

Интересная задача – полное управление, мониторинг, контроль внутренней сети предприятия без IP-доступа (без подключения), с помощью локального ИИ. Никаких сессий. При этом можно не только управлять сетью или серверами, но и получать информацию по запросам любой сложности, создавать скрипты, делать выборки из справочников, почты и баз данных и т.д.

https://proposed-gray-cattle.myfilebase.com/ipfs/QmeU3EYvwA3HDnRwyB2xUAfD39WQgnF9WfimVwumw6s5Ba

https://proposed-gray-cattle.myfilebase.com/ipfs/Qmdjn3PA14PLyzmz1kwFfcmQnZRxBgYvcE8qcU2kAX86gi

Head Hunter

Вообще, тема создания информационных каналов «для тяжёлых ограничений» - крайне интересная задача. Конечно, не смотря на множество путей передачи информации в IPFS, его тоже можно блокировать. В таком случае можно использовать несколько протоколов одновременно. Например IPFS+Bit Torrent и т.д.

В случае ещё больших ограничений можно создавать временные копии сайтов для просмотра. За периметром ограничений создаётся специализированный сервер, делающий статическую «одноразовую» копию необходимой веб страницы, которую можно просматривать по совершенно не связанному с оригиналом адресу, либо архивировать ее и переслать по почте.

Технически возможно написать код и обеспечить интерактивность, но это задача не простая. Здесь явно напрашивается использование API для работы с Искусственным Интеллектом. Т.е. это задача создания «разумного» Proxy-сервера (с API или изолированный AI под Ollama). Ничего невозможного в этом нет. Образцы уже разработаны и тестируются.

Ещё один вариант преодоления ограничений – использование ПО на VPS, работающего с кастомными сетевыми пакетами. Т.е. тут уже идёт речь о создании собственных алгоритмов и протоколов установления подключений и передачи данных. Опять же, с использованием ИИ API это решаемая задача.

Head Hunter

Для генерирования Python-кода IPFS прокси сервера с созданием статической версии веб страницы разработан Prompt. Использование этого Prompt позволяет очень быстро дорабатывать Python код под новые требования и ограничения и даже перейти на другой язык. Prompt можно использовать в Qwen, DeepSeek, ChatGPT и т.д.

Реальный сгенерированный Python код работает на VPS под Ubuntu

Prompt для генерации Proxy2 IPFS Snapshot Server

1. **Общее описание задачи**

Создай Python-скрипт proxy2_ipfs_snapshot.py для сервера, который:

Принимает HTTP-запросы через IPFS P2P канал (от Proxy1 на ПК пользователя)

Определяет запросы на создание снэпшота по параметру ?snapshot=URL

Загружает целевую страницу через headless-браузер с выполнением JavaScript

Сохраняет все ресурсы (картинки, CSS, шрифты) в IPFS

Подменяет ссылки в HTML на IPFS CID

Очищает HTML от JavaScript и обработчиков событий

Возвращает готовую HTML-страницу браузеру пользователя

2. **Технологический стек (с обоснованием)**

Компонент	Технология	Причина выбора
Рендеринг JS	Playwright	Поддерживает все современные JS-фреймворки, имеет API для перехвата ресурсов, встроенная защита от обнаружения ботов
Парсинг HTML	BeautifulSoup4	Легковесный, надёжный, не ломает структуру документа
IPFS интеграция	ipfshttpclient	Официальный клиент, поддерживает add_bytes() и cat()
Кодирование HEX	binascii	Встроенный модуль, быстрое преобразование для передачи через IPFS
Многопоточность	threading	Встроенный модуль, достаточен для одновременных соединений
Очистка от JS	BeautifulSoup + re	Удаление тегов <script> и атрибутов on*

3. **Глобальные переменные и конфигурация**

python

```
LISTEN_PORT = 8222                # Порт для приёма соединений от IPFS (согласован с ipfs p2p)
EXTERNAL_IP = "192.168.5.150"     # Внешний IP для привязки сокета (IP вашей VMware/хостинга)

ENABLE_SNAPSHOT_MODE = True       # Включить режим создания снэпшотов
MAX_RESOURCE_SIZE = 50 * 1024 * 1024  # 50 MB - максимальный размер скачиваемого ресурса
PAGE_LOAD_TIMEOUT = 30            # 30 секунд на загрузку страницы

# Префикс для HEX-кодирования (нужен для совместимости с Proxy1)
# Используем существующий GET-запрос как маркер начала данных
HEX_PREFIX = b"GET /wiki/ HTTP/1.1\r\nHost: eu.euwiki.io\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36\r\nAccept: text/html\r\nAccept-Language: ru-RU,ru;q=0.9\r\nAccept-Encoding: gzip, deflate, br\r\nConnection: keep-alive\r\n\r\n"
SEPARATOR = b"\n---END---\n"      # Разделитель между префиксом и HEX-данными

ipfs_client = None                # Глобальный объект IPFS клиента (инициализируется в main)
```

4. **Функции (по порядку)**

4.1. **IPFS функции**

python

def init_ipfs_client() -> bool

Назначение: Инициализирует соединение с локальным IPFS демоном

Параметры: нет

Возвращает: True если успешно, False если ошибка

Использует: ipfshttpclient.connect('/dns/localhost/tcp/5001/http')

Обработка ошибок: Перехватывает ImportError и Exception

python

def upload_to_ipfs(data: bytes) -> str | None

Назначение: Загружает бинарные данные в IPFS

Параметры: data - байты для загрузки

Возвращает: CID в виде строки или None при ошибке

Использует: ipfs_client.add_bytes(data)

Примечание: CID используется для подмены ссылок на /ipfs/{cid}

4.2. **Функции рендеринга**

python

def handle_response(response) -> None

Назначение: Callback для перехвата всех ответов браузера (внутри render_and_snapshot)

Параметры: response - объект ответа Playwright

Что делает:

Фильтрует по типу: image, stylesheet, font

Пропускает data: URI

Проверяет размер (не больше MAX_RESOURCE_SIZE)

Загружает в IPFS через upload_to_ipfs()

Сохраняет маппинг url -> cid в словаре resources

python

def replace_urls_with_ipfs(html: str, base_url: str, resources: dict) -> str

Назначение: Подменяет ссылки на ресурсы в HTML

Параметры:

html - исходный HTML

base_url - базовый URL страницы (для преобразования относительных ссылок)

resources - словарь {original_url: cid}

Возвращает: Изменённый HTML

Использует: BeautifulSoup(html, 'html.parser')

Что делает:

Заменяет src у всех <img> на /ipfs/{cid}

Заменяет href у <link rel="stylesheet">

Удаляет все <script> теги

Удаляет все атрибуты начинающиеся с on (onclick, onload и т.д.)

Добавляет мета-тег <meta name="generator" content="IPFS-HTTP-Snapshot">

python

def render_and_snapshot(url: str) -> bytes

Назначение: Основная функция рендеринга страницы

Параметры: url - целевой URL для снэпшота

Возвращает: Полный HTTP-ответ с HTML (в байтах)

Использует: Playwright sync_api

Алгоритм:

Запуск headless Chromium с анти-детект аргументами

Создание контекста с размером окна 1920x1080

Регистрация handle_response как обработчика

Переход по URL с ожиданием networkidle

Прокрутка страницы до конца для lazy-loading

Получение page.content()

Вызов replace_urls_with_ipfs()

Формирование HTTP-ответа через html_response()

python

def html_response(html: str) -> bytes

Назначение: Формирует корректный HTTP-ответ для браузера

Параметры: html - содержимое HTML

Возвращает: HTTP/1.1 200 OK с заголовками

Заголовки:

Content-Type: text/html; charset=utf-8

Content-Length

X-Snapshot: true

Cache-Control: no-cache

python

def error_response(message: str) -> bytes

Назначение: Формирует страницу с ошибкой

Параметры: message - текст ошибки

Возвращает: HTML-страницу с сообщением об ошибке

4.3. **Функции обработки запросов**

python

def parse_snapshot_request(data: bytes) -> str | None

Назначение: Определяет, является ли запрос запросом снэпшота

Параметры: data - сырые байты запроса

Возвращает: Целевой URL или None

Поддерживаемые форматы:

GET /?snapshot=https://example.com

GET /http://example.com/

python

def extract_hex_data(data: bytes) -> bytes

Назначение: Декодирует HEX-данные из пакета (для совместимости с Proxy1)

Параметры: data - закодированные данные с префиксом

Возвращает: Декодированные байты

Алгоритм:

Удаляет HEX_PREFIX и SEPARATOR

Разбивает по строкам

Преобразует каждую строку через binascii.unhexlify()

python

def encode_data(data: bytes) -> bytes

Назначение: Кодирует данные в HEX с префиксом (для обратной совместимости)

Параметры: data - сырые байты

Возвращает: HEX_PREFIX + SEPARATOR + hex(data)

4.4. **Основная логика**

python

def handle_tunnel(client_sock: socket.socket, addr: tuple) -> None

Назначение: Обрабатывает одно P2P соединение

Параметры:

client_sock - сокет клиента (от IPFS)

addr - адрес клиента

Алгоритм:

Получение данных от клиента

Декодирование через extract_hex_data()

Проверка на запрос снэпшота через parse_snapshot_request()

Если снэпшот - вызов render_and_snapshot()

Кодирование ответа через encode_data()

Отправка клиенту

python

def main() -> None

Назначение: Точка входа

Алгоритм:

Вывод информации о запуске

Инициализация IPFS клиента

Запуск ipfs p2p listen как дочернего процесса

Создание TCP сервера на 127.0.0.1:LISTEN_PORT

Бесконечный цикл приёма соединений

Создание потока для каждого соединения

5. **Аргументы запуска браузера (анти-детект)**

python

```
args=[
    '--disable-blink-features=AutomationControlled',  # Отключает флаг автоматизации
    '--disable-dev-shm-usage',                  # Для работы в Docker/VM
    '--no-sandbox'                          # Для Ubuntu без sandbox
```

]

6. **Инструкция по установке и запуску**

На Ubuntu (хостинг/VM с Proxy2)

bash

1. **Обновление системы**

sudo apt update && sudo apt upgrade -y

2. **Установка Python и pip**

sudo apt install python3 python3-pip python3-venv -y

3. **Установка зависимостей Python**

pip3 install playwright beautifulsoup4 ipfshttpclient

4. **Установка браузера для Playwright**

playwright install chromium

5. **Проверка и запуск IPFS демона**

ipfs init # если не инициализирован

ipfs daemon &

6. Запуск Proxy2

python3 proxy2_ipfs_snapshot.py

Настройка IPFS

bash

Если IPFS не установлен:

wget https://dist.ipfs.tech/kubo/v0.28.0/kubo_v0.28.0_linux-amd64.tar.gz

tar -xvzf kubo_v0.28.0_linux-amd64.tar.gz

cd kubo

sudo bash install.sh

Настройка CORS для IPFS (если нужно)

ipfs config --json API.HTTPHeaders.Access-Control-Allow-Origin '["*"]'

ipfs config --json API.HTTPHeaders.Access-Control-Allow-Methods '["GET", "POST", "PUT"]'

Проверка работы

bash

Должны увидеть:

==

PROXY2 - IPFS Snapshot Server

==

[*] External IP: 192.168.1.160

[*] Listening port: 8222

[✓] IPFS client connected

[*] Starting: ipfs p2p listen /x/http-proxy/1.0.0 /ip4/127.0.0.1/tcp/8222

[*] Waiting for connections...

7. **Формат запроса от Proxy1**

Proxy1 (на ПК пользователя) должен отправлять закодированные запросы:

text

GET /?snapshot=https://news.ru/article/123 HTTP/1.1

Host: localhost

В закодированном виде (HEX + префикс):

text

{HEX_PREFIX}---END---{HEX_DUMP}\n

8. **Ограничения и известные проблемы**

Проблема	Решение
Некоторые сайты блокируют headless-браузеры	Используются аргументы --disable-blink-features
Большие страницы (много картинок)	Ограничение MAX_RESOURCE_SIZE = 50 MB
Lazy-loading изображений	Прокрутка страницы до конца + 2 секунды ожидания
Шрифты из CSS	Поддерживаются через inline замену URL
Сайты с бесконечным скроллом	Не поддерживаются (снэпшот только видимой части)

9. **Рекомендации по дальнейшему улучшению**

Кэширование CID - сохранять соответствие URL→CID на диск для повторного использования

Сжатие изображений - использовать PIL для уменьшения размера перед загрузкой в IPFS

Асинхронность - переписать на asyncio + async playwright для большей производительности

Поддержка WebSocket - игнорировать или логировать

Авторизация - добавление поддержки cookies/сессий для сайтов с авторизацией

10. **Полный код должен быть**

Самодостаточным (одним файлом)

С подробными комментариями на русском или английском

С обработкой всех исключений

С выводом в консоль для отладки

Совместимым с Python 3.8+

Сгенерируй полный код скрипта proxy2_ipfs_snapshot.py согласно этому промпту.

Prompt для генерации Proxy2 IPFS Snapshot Server

https://oflameron.com

Head Hunter